EXHIBIT BB

ZATARAIN II

FIRST SUPPLEMENT TO THE FEBRUARY 24, 2012 OPENING EXPERT REPORT OF ARTHUR ZATARAIN, PE

MARCH 2, 2012

IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF WISCONSIN

ROCKWELL AUTOMATION, INC. and ROCKWELL AUTOMATION TECHNOLOGIES, INC., Plaintiffs, v. WAGO CORPORATION and WAGO KONTAKTTECHNIK GmbH & CO. KG, Defendants.	Case No. 10-CV-718-WMC

FIRST SUPPLEMENT TO THE FEBRUARY 24, 2012 EXPERT REPORT OF ARTHUR ZATARAIN ON BEHALF OF PLAINTIFFS ROCKWELL AUTOMATION, INC. AND ROCKWELL AUTOMATION TECHNOLOGIES, INC.

415. I, Arthur Zatarain, have been retained by Plaintiffs Rockwell Automation, Inc. and Rockwell Automation Technologies, Inc. ("Rockwell") as a technical expert consultant and potential testifying expert witness in this lawsuit filed against WAGO Corporation and WAGO Kontakttechnik GmbH & Co. KG. I am the same Arthur Zatarain who submitted the Expert Report that was served on February 24, 2012 (the "Report"). I submit this supplement in support of my Report. The paragraphs in this supplement are numbered to follow in conjunction with or in sequence with my Report.

416. Unless stated otherwise, the definitions from my Report apply herein.

417. In Exhibit B to my Report I set forth a List of Documents Reviewed. In addition to those materials, I have read the following materials:

a. WKT 041644 - WKT 042177

3. WAGO's Infringement of U.S. Patent No. 6,801,813

p. Claim 20

354. Claim 20 reads as follows:

> 20. The system of claim 1, the industrial control program being a ladder logic program.

355. Claim 20 includes the element "the industrial control program being a ladder logic program." The industrial control program stored on the Infringing File System PLCs contains ladder logic instructions.

356. Claim 20 is met literally, or in the alternative, under the doctrine of equivalents by the Infringing File System PLCs for at least the following reasons:

a. WAGO's 30(b)(6) witness agreed with the assertion that a WAGO customer could write a program "containing Ladder Logic instructions to employ the file system on the WAGO industrial controller to log data to a file." (See Albers 30(b)(6) Dep. Tr. 84:22-85:11).

b. My testing confirmed that the infringing File System PLC has ladder logic instructions to employ file system services to both log and read data to a file.

c. In the event that claim interpretation by the Court should result in no literal infringement, it is my view that any reasonable interpretation of this claim element would still result in infringement under the doctrine of equivalents since the industrial control program stored on the Infringing File System PLCs contains ladder logic instructions as set forth herein, and would thus perform substantially the same function in substantially the same way to yield substantially the same result.

357. Based on the foregoing, and based on other information disclosed in this report, it is my opinion that WAGO has infringed and is infringing Claim 20 of the '813 patent literally, or in the alternative, under the doctrine of equivalents.

4. WAGO's Infringement of U.S. Patent No. 7,065,415

391. The asserted claims of the '415 patent are directed to "An editor for developing ladder logic programs that control operation of an industrial controller system, the editor comprising". It is my opinion that the WAGO-I/O System provides an editor for developing ladder logic programs that control operation of an industrial controller system based on WAGO's 30(b)(6) deponent, Dr. Thomas Albers, admission of infringement of each and every element of the asserted claims of the '415 patent. (See Albers 30(b)(6) Dep. 84:22-85:4; 85:6-11; 77:21-80:12; 85:13-86:15; 84:22-85:4, 91:1-8; 68:1-4; 80:25-81:3; 78:8-12:80-12; 86:5-15; 142:7-12). This was confirmed in my testing as noted in the attached Supplemental Test Report.

.

Dated: March 2, 2012

Arthur Zatarain, PE

CERTIFICATE OF SERVICE

I hereby certify that on this 2nd day of March, 2012, pursuant to a written agreement between the parties' counsel governing service in this action, a copy of this document is being served by electronic mail on Defendants' counsel listed below at the email addresses listed below:

Robert Cook, Esq.
Whitham, Curtis, Christofferson & Cook, P.C.
11491 Sunset Hills Road, Suite 340
Reston, Virginia 20190
Tel.: (703) 787-9400, ext. 122
Fax: (703) 787-7557
Email: bob@wcc-ip.com

John C. Scheller, Esq. (SBN 1031247)
Michael Best & Friedrich LLP
One South Pinckney Street – Suite 700
Madison, WI 53703
Tel.: (608) 257-3501
Fax: (608) 283-2275
Email: jcscheller@michaelbest.com

/s/ Paul Tanck
Paul Tanck

Wago I/O Supplemental Test Notes

March 2012

1. This report was prepared by Arthur Zatarain, Artzat Consulting, Metairie LA. Contact info 504-837-3090, artzat.com.

2. The Artzat project number was 2012-324. Testing was conducted at the office of Artzat Consulting in Metairie, LA. All testing and documentation was prepared by Arthur Zatarain.

3. This report provides test notes and related information on supplemental testing conducted in March 2012 on a WAGO 758-870-130 PLC. The tested PLC was made available to me following my initial report was submitted and could not be included in the original report.

4. The WAGO PLC is identified and was configured as follows:

   4.1. I/O-IPC-G2

      4.1.1. Unit marked with 758-870/0000-0130, s/n 188640007, HW 10 Rev 12, 27.01.2011

      4.1.2. Unit has CoDeSys holographic sticker on the bottom, SNR 100411750

      4.1.3. Fitted with 750-600 8 point digital I/O slice addressed as IX0.0-0.8 and QX0.0-0.8.

      4.1.4. Network connection to port X8 at 192.168.1.17.

      4.1.5. Alt-F2 gets to command console, log in to root with password=wago.

      4.1.6. Alt-F1 gets to VGA screen from Codesys. Alt-F3 gets to config screen.

      4.1.7. CoDeSys runtime plclinux_rt was located in dir /usr/sbin. The file is approx. 1.7Mb, dated 9/27/2011.

      4.1.8. Codesys working dir seems to be /home/codesys.

      4.1.9. External USB drive is mounted by Linux at /media/sda1.

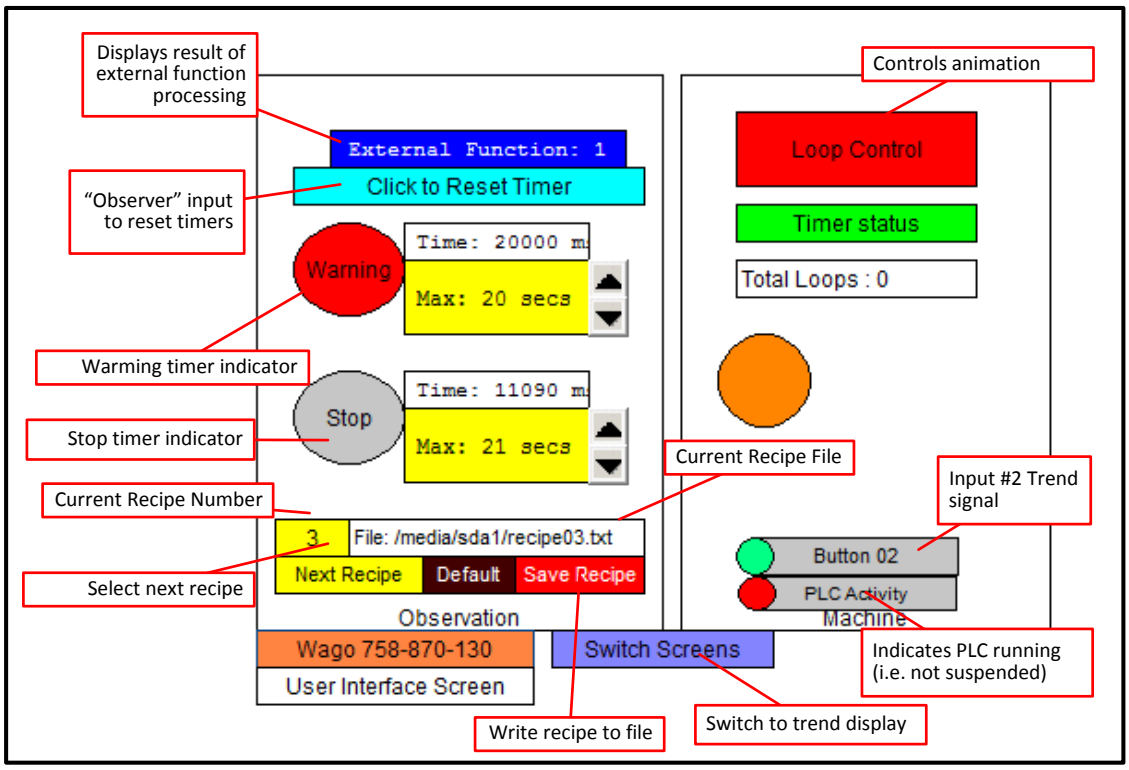
      4.1.10. Attached I/O was an 8 input / 8 output module.

5. Test Procedure

   5.1. The same computer and network used in earlier testing was used to test the 758-870-130 IPC. The original demonstration program was modified to use the -130 variant as the target PLC. The program was also modified to use ladder logic instructions for file read, file write, and file close. Other program features including the use of a loadable external user routine were left unchanged.

   5.2. A CoDeSys project documentation printout is included at the end of this report.

   5.3. The demonstration program functioned in the same manner as the program tested on the -110 variant of the 870 PLC. A screenshot of the -130 program interface is shown below:



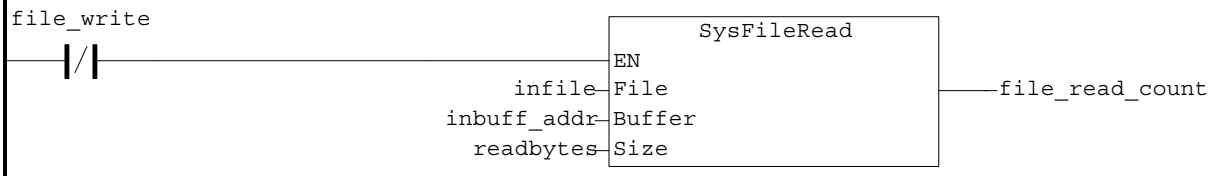
Demonstration Interface Screen for 758-870-130 IPC

Filename:	Wagotest_05.pro
Directory:	C:\Wagotest\Wagotest_758_870_130_2
Change date:	2.3.12 12:20:10 / V2.3
Title:	Wago PLC demonstration
Author:	Arthur Zatarain
Version:	1.0
Description:	Demo package in Wago Pro CAA.

```
0001 PROGRAM file_io_1d
0002 VAR
0003 END_VAR
```

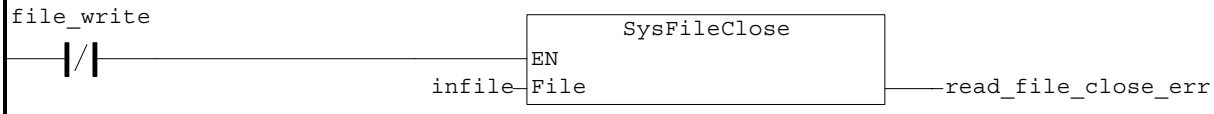
0001

Read data from file into buffer



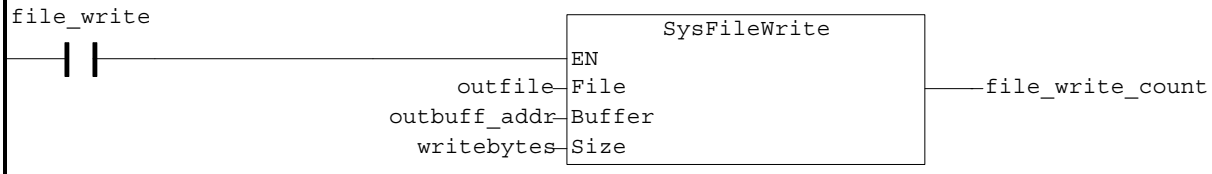
0002

Close file after reading



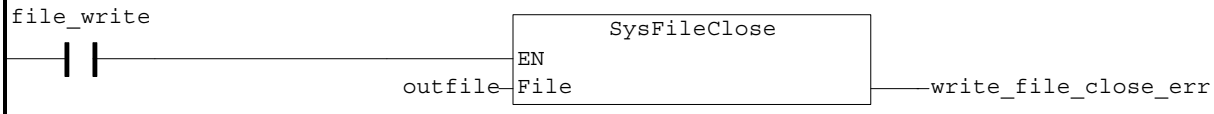
0003

Write data to file



0004

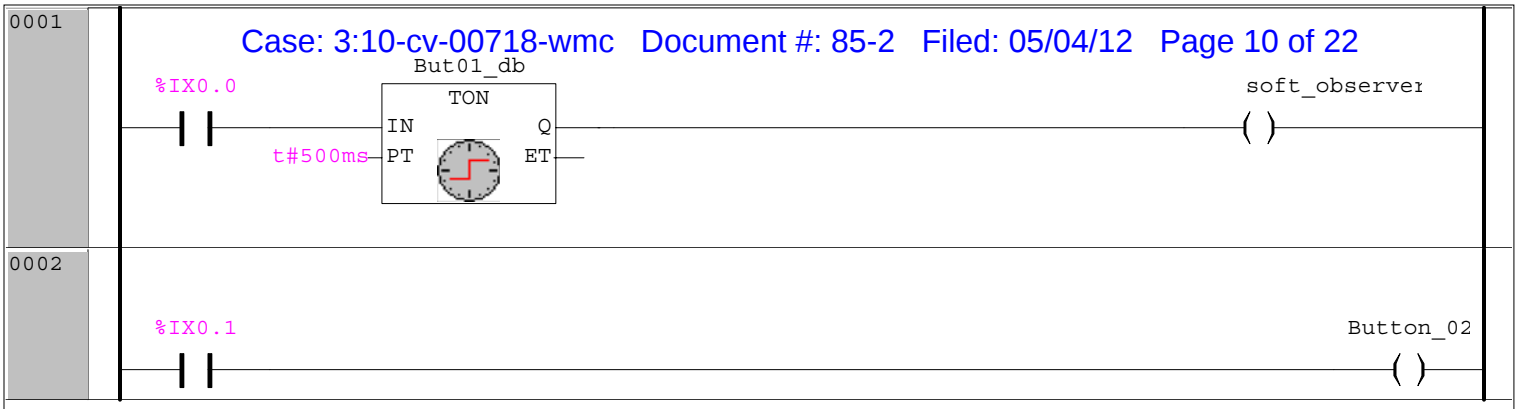
Close file after writing



0005

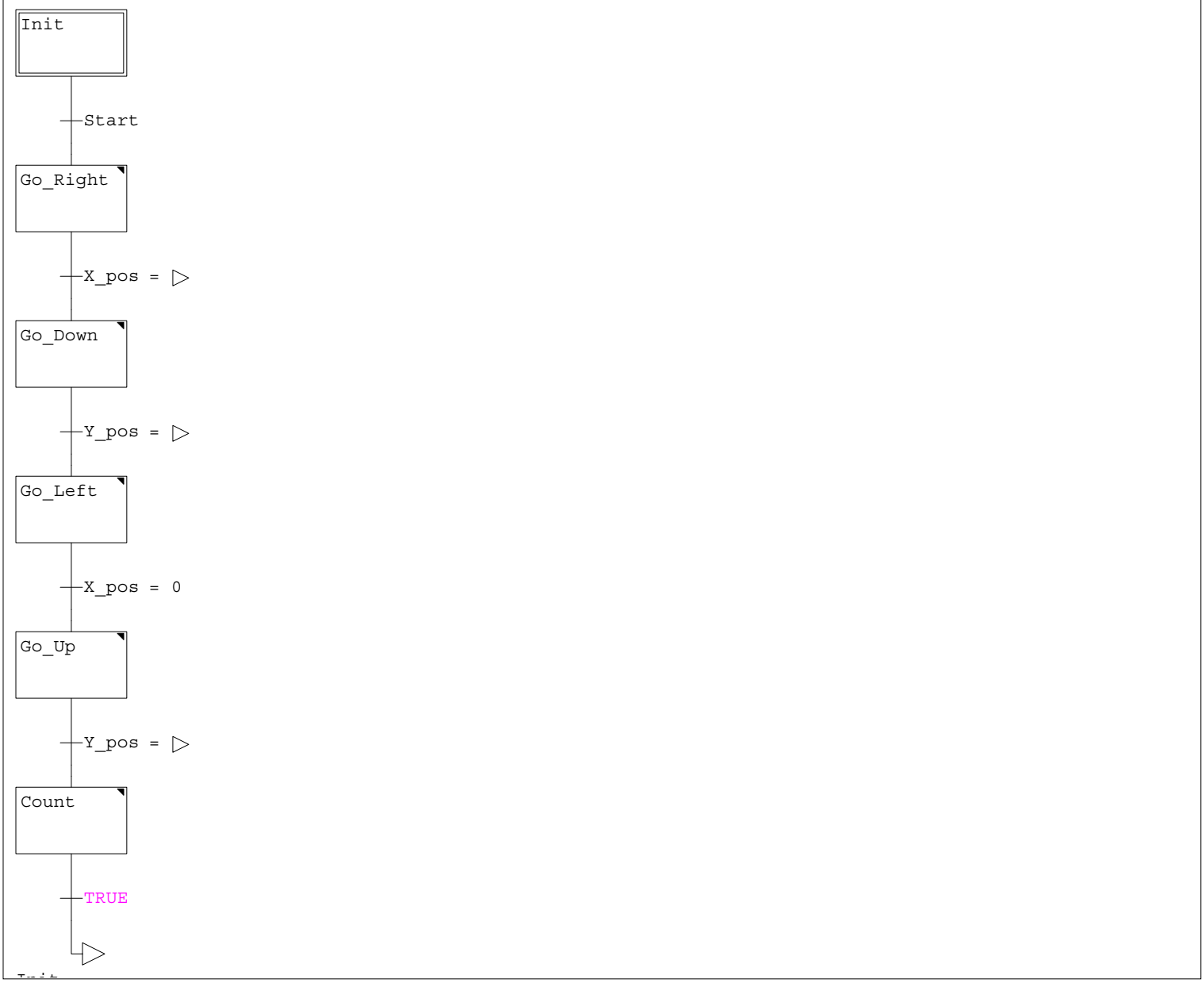
input_switches (PRG-LD)

```
0001 (* AMZ Feb 13 2012
0002
0003 Demo program to read physical input switches to activate
0004 soft equivalent within program.
0005 *)
0006
0007 PROGRAM input_switches
0008 VAR
0009     Button_02: BOOL;
0010     But01_db: TON;
0011     soft_observer: BOOL;
0012     But02_db: TON;
0013 END_VAR
```



Machine (PRG-SFC)

```
0001  (* AMZ Feb 13 2012
0002
0003  Demonstration of screen animation adopted from Wago/CoDeSys demo.
0004  *)
0005
0006  PROGRAM Machine
0007  VAR
0008      X_pos: INT;
0009      Y_pos: INT;
0010      Counter: INT;
0011  END_VAR
```



Init

Machine (PRG-SFC).Aktion Go_Right (ST)

```
0001 X_pos := X_pos + 1 ;
```

Machine (PRG-SFC).Action Go_Down (ST)

```
0001 Y_pos := Y_pos + 1 ;
```

Machine (PRG-SFC).Aktion Go_Left (ST)

```
0001 X_pos := X_pos - 1 ;
```

Machine (PRG-SFC).Action Go_Up (ST)

```
0001 Y_pos := Y_pos - 1 ;
```

Machine (PRG-SFC).Aktion Count (ST)

```
0001 Counter := Counter + 1 ;
0002
0003
0004
0005
0006
```

modify_delay (FUN-ST)

```
0001 (* AMZ Feb 13 2012
0002
0003 Demo program to process screen buttons to modify alarm and warning
0004 timer values.
0005 *)
0006
0007 FUNCTION modify_delay : BOOL
0008 VAR_INPUT
0009   delay_number : WORD;  (* which delay to modify *)
0010   delay_change : INT; (* variance *)
0011 END_VAR
0012 VAR
0013   tempw : WORD;
0014   temp_delay : INT;
0015   temp_secs : WORD;
0016
0017 END_VAR
```

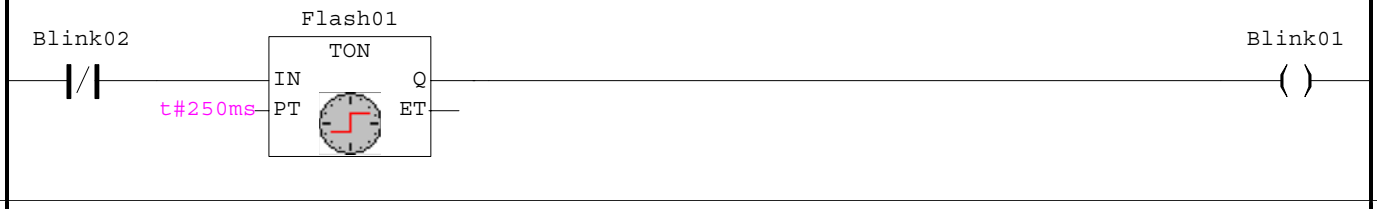
```
0001 (* change delay time in seconds based on passed incremental value *)
0002
0003 modify_delay := FALSE;
0004
0005 CASE delay_number OF
0006     1 : temp_delay := TIME_TO_INT(delay01);
0007     2 : temp_delay := TIME_TO_INT(delay02);
0008 END_CASE
0009
0010 temp_delay := temp_delay / 1000;  (* convert ms to seconds *)
0011 temp_delay := temp_delay + delay_change;  (* convert in whole seconds *)
0012
0013 IF temp_delay < 5 THEN temp_delay := 5; END_IF
0014 IF temp_delay > 99 THEN temp_delay := 99; END_IF
0015 temp_secs := temp_delay;  (* delay in seconds *)
0016
0017 temp_delay := temp_delay * 1000;  (* back to ms *)
0018
0019 CASE delay_number OF
0020   1 : delay01 := INT_TO_TIME(temp_delay); delay01_secs := temp_secs;
0021   2 : delay02 := INT_TO_TIME(temp_delay); delay02_secs := temp_secs;
0022 END_CASE
0023
0024 modify_delay := TRUE;  (* return valu *)
0025
```

Output_LED (PRG-LD)

```
0001 (* AMZ Feb 13 2012
0002
0003 Ladder logic to provide global flash timer.
0004 *)
0005
0006 PROGRAM Output_LED
```

```
0007 VAR
0008     Flash01: TON;
0009     Flash02: TON;
0010     Blink01: BOOL;
0011     Blink02: BOOL;
0012 END_VAR
```

0001



0002

Bool to control flashing

0003

0004

Normal run light

0005

Warning lightt

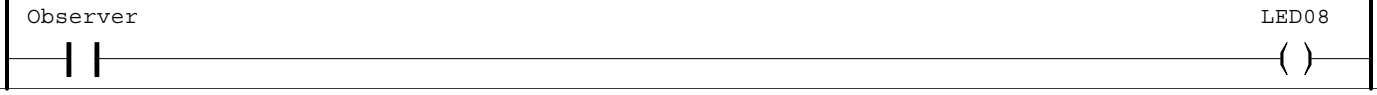
0006

Failed light

0007

0008

0009

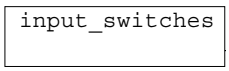


PLC_PRG (PRG-FBD)

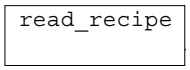
```
0001 (*  AMZ Feb 13 2012
0002
0003 Main POU for Wago PLC demonstration.
```

```
0004 This unit is called on program start and schedules all
0005 other units.
0006
0007 *)
0008
0009 PROGRAM PLC_PRG
0010 VAR
0011     Trig1: R_TRIG;
0012     Trig2: F_TRIG;
0013     Timer1: TOF;
0014     Timer2: TON;
0015     Timer3: TP;
0016     trig_observer: BOOL;
0017 END_VAR
```

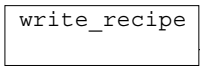
0001



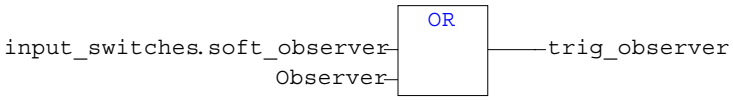
0002



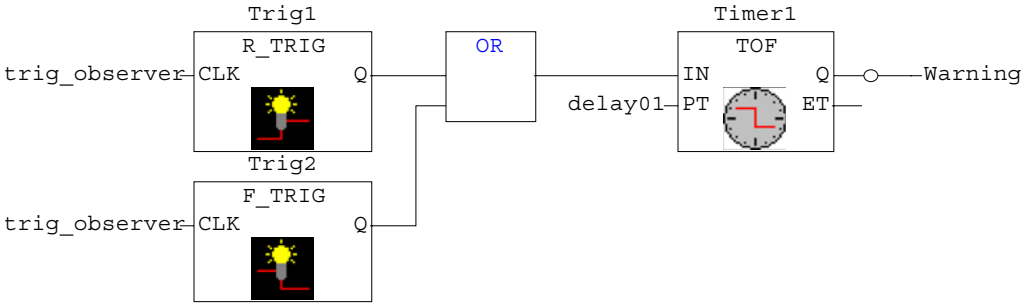
0003



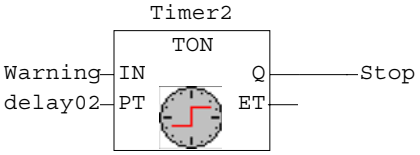
0004



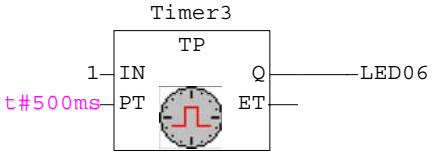
0005



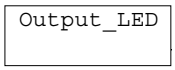
0006



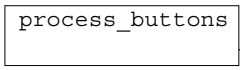
0007



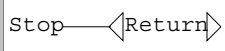
0008 Update the PLC indicator LEDs



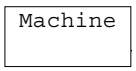
0009 Proces the functions related to button presses



0010 do not continue if stopped



0011 Operate state machine when not stopped.



process_buttons (PRG-ST)

```
0001 PROGRAM process_buttons
0002 VAR
0003
0004 END_VAR
```

```
0001 (*  AMZ Feb 13 2012
0002     demo program button processor *)
0003
0004 (* increment passed value on first transition of the
0005 variable observer. The increment will depend on which linkable
0006 routine is present on the controller during runtime.
0007
0008 All versions of the demo function are called by the same
0009 function name MyTestFunction
0010 *)
0011
0012 IF observer   THEN
0013   demo_val := MyTestFunction(demo_val);
0014 (*  count01 := TRUE; *)
0015   observer := FALSE;
0016 END_IF;
0017
0018
0019 (* process buttons for delay value change. *)
0020
0021 IF del01u THEN
0022   modify_delay(1, 1);  (* timer 1 up 1 *)
0023   del01u := FALSE;
0024 END_IF
0025
0026 IF del01d THEN
0027   modify_delay(1, -1);  (* timer 1 up 1 *)
0028   del01d := FALSE;
0029 END_IF
0030
0031 IF del02u THEN
0032   modify_delay(2, 1);  (* timer 1 up 1 *)
0033   del02u := FALSE;
0034 END_IF
0035
0036 IF del02d THEN
0037   modify_delay(2, -1);  (* timer 1 up 1 *)
0038   del02d := FALSE;
0039 END_IF
```

```
PROGRAM read_recipe
VAR
    fname : STRING := 'none';


    tempw: WORD;
    new_recipe : INT;
    file_read: BOOL;
END_VAR
VAR_OUTPUT
END_VAR
```

```
(* AMZ Mar 2 2012
Demonstration of Wago PLC program to read recipe information from a text file
using the file services on the PLC.

Four files can be read as determined by variable cur_recipe.

The text file has 5 signifiant characters. First 2 are ascii decimal of
warning delay.Then a space (or comma), then the stop delay.
Minimum delay is 5 seconds, max is 99.
*)

IF read_recipe_button THEN
      IF curr_recipe < 1 THEN
        new_recipe := 1;  (* try to load the first recipe file *)
      ELSE
        new_recipe := curr_recipe+1; (* next recipe file *)
      END_IF
     IF new_recipe > 4 THEN new_recipe := 1; END_IF  (* max of 4 recipes *)

    CASE new_recipe OF
        1: fname := rec_fname_01;
        2: fname := rec_fname_02;
        3: fname := rec_fname_03;
        4: fname := rec_fname_04;
        ELSE fname := rec_fname_01;
    END_CASE

    infile := 0;  (* clear file handle *)
    infile := SysFileOpen(fname, 'r');  (* try to open the file *)
    file_read_count := 0;  (* assume no bytes will be read *)
    IF (infile >0) THEN  (* file opend ok *)
        inbuff_addr := ADR(inbuffer);
        file_write := FALSE; (* signal io system to read from file *)
        file_io_ld;  (* use ladder instructions to read data into buffer *)

    ELSE
         load_def_recipe := TRUE;  (* set defaults if no file read *)
    END_IF
    infile := 0;  (* clear file handle *)

      IF (file_read_count >= readbytes) THEN    (* convert first 2 characters in the
                                                 array to delay0 word *)
(* remove ascii offset 48 *)
        tempw := ((inbuffer[0] - 48) * 10) + (inbuffer[1] - 48);
        IF tempw < 5 THEN tempw := 5; END_IF
        delay01_secs := tempw;
        tempw := tempw * 1000;  (* convert to milliseconds *)
        delay01 := WORD_TO_TIME(tempw);  (* warming timer *)
(* skip over separator at [2] and get next 2 digits *)
        tempw := ((inbuffer[3] -48) * 10) + (inbuffer[4] - 48);
        IF tempw < 5 THEN tempw := 5; END_IF
        delay02_secs := tempw;
        tempw := tempw * 1000;  (* convert to milliseconds *)
        delay02 := WORD_TO_TIME(tempw);  (* stop timer *)
        curr_recipe := New_recipe;
    END_IF

    read_recipe_button := FALSE;
```

```
0059 END_IF
0060
0061 IF load_def_recipe THEN
0062     new_recipe := 0;  curr_recipe:= 0;  (* use defaults *)
0063     fname := 'None - using default';
0064     delay01 := default_delay; delay02 := default_delay;
0065     delay01_secs := TIME_TO_WORD(default_delay / 1000);  (* delay in seconds *)
0066     delay02_secs := delay01_secs;
0067     load_def_recipe := FALSE;
0068     read_recipe_button := FALSE;
0069 END_IF
0070
```

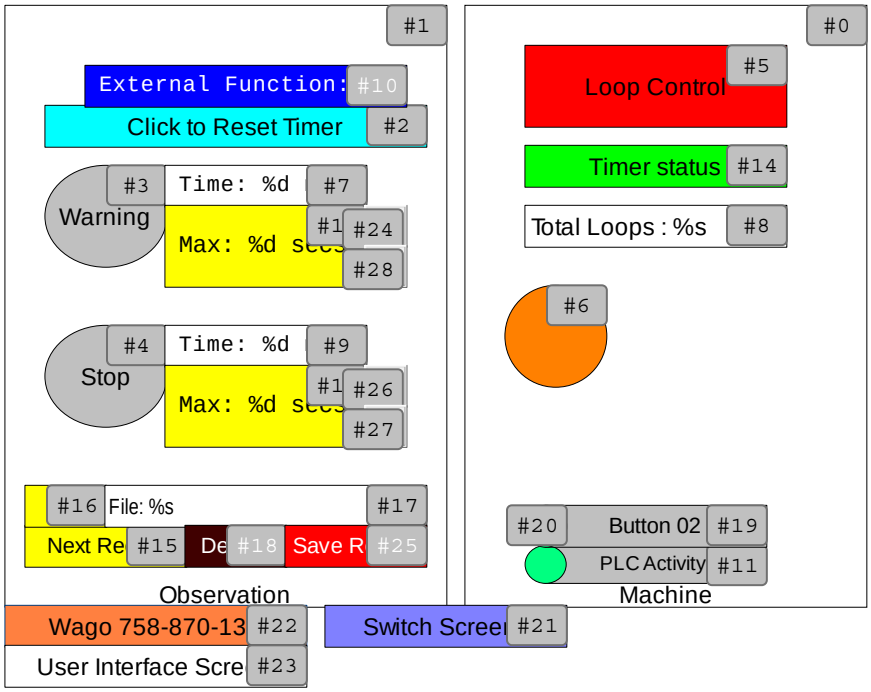
write_recipe (PRG-ST)

```
0001 PROGRAM write_recipe
0002 VAR
0003     fname : STRING := 'none';
0004
0005     tempw: WORD;
0006     save_recipe : INT;
0007     ii : WORD;
0008
0009     write_file: BOOL;
0010 END_VAR
0011 VAR_OUTPUT
0012 END_VAR
```

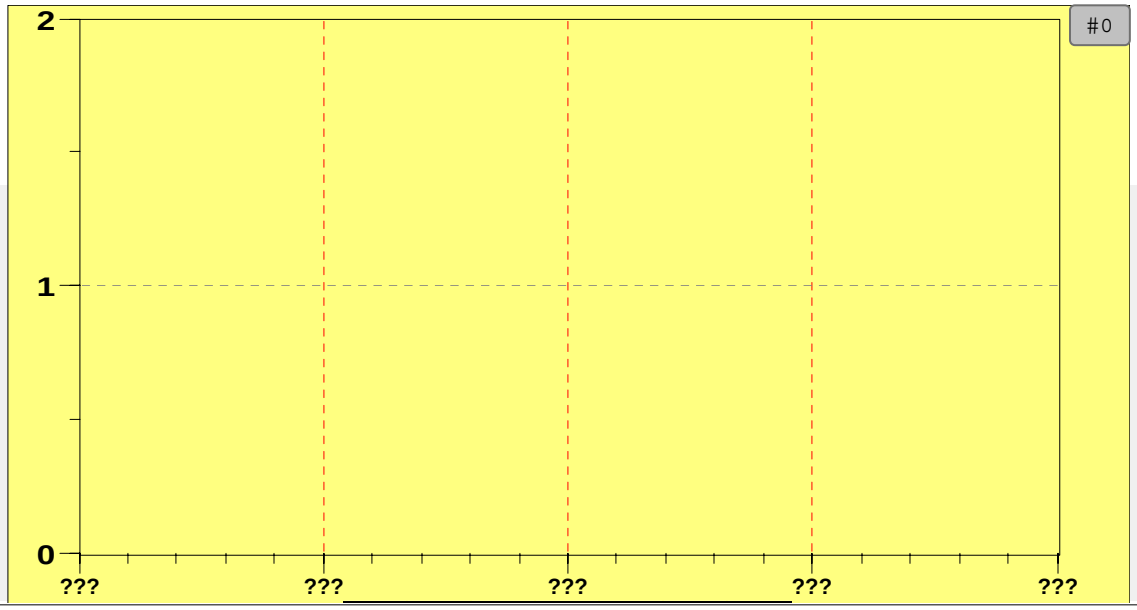
```
0001 (* AMZ Feb 13 2012
0002 demonstration of Wago PLC program to write date into a file using
0003 the file services on the PLC. The files to be written is determined by
0004 variable curr_recipe. The file has two 2-characer values in a simple
0005 text file in the format ABxCD.  AB is the 2 character representation
0006 of the warning delay in seconds. The middle character x can be anything,
0007 and a space is used in this example.
0008 The final pair CD is the 2 char representation of the alarm time in seconds.
0009 *)
0010
0011 IF write_recipe_button THEN
0012     save_recipe := curr_recipe; (* local of current global recipe *)
0013     IF ((save_recipe <1) OR (save_recipe > 4)) THEN
0014         load_def_recipe := TRUE;
0015         read_recipe; (* use default values *)
0016         write_recipe_button := FALSE;
0017     END_IF
0018 END_IF
0019
0020 IF write_recipe_button THEN  (* still OK to write *)
0021     save_recipe := curr_recipe;  (* max of 4 recipes *)
0022     CASE save_recipe OF
0023         1: fname := rec_fname_01;
0024         2: fname := rec_fname_02;
0025         3: fname := rec_fname_03;
0026         4: fname := rec_fname_04;
0027     END_CASE
0028
0029 (* put space characters into the array to delay0 word *)
0030     FOR ii := 0 TO 9 DO
0031       outbuffer[ii] := 32;  (* ascii space *)
0032     END_FOR
0033     tempw := TIME_TO_WORD(delay01) / 1000; (* milliseconds to secs *)
0034     IF tempw < 5 THEN tempw := 5; END_IF  (* min time allowed *)
0035     IF tempw > 99 THEN tempw := 99;  END_IF (* max delay secs *)
0036     IF tempw > 9 THEN  (* need 2 digits *)
0037       outbuffer[0] := WORD_TO_BYTE(tempw / 10) + 48;  (* upper digit *)
0038     ELSE
0039       outbuffer[0] := 48;  (* upper digit is ascii 0*)
0040     END_IF
0041     outbuffer[1] := WORD_TO_BYTE(tempw MOD 10) + 48;  (* lower *)
0042
0043     tempw := TIME_TO_WORD(delay02) / 1000;
0044     IF tempw > 99 THEN tempw := 99; END_IF
0045     IF tempw > 9 THEN
```

```
0046        outbuffer[3] := WORD_TO_BYTE(tempw / 10) + 48;  (* upper digit *)
0047      ELSE
0048        outbuffer[3] := 48;  (* upper digit is ascii 0*)
0049      END_IF
0050      outbuffer[4] := WORD_TO_BYTE(tempw MOD 10) + 48;  (* lower *)
0051      outbuffer[5] := 16#0A; outbuffer[6] := 16#0D;  (* cr/lf pair *)
0052
0053  (* write buffer to file *)
0054      outfile := SysFileOpen(fname, 'w');  (* try to open the file *)
0055      writebytes := 10;  (* write all 10 bytes *)
0056
0057      IF (outfile > 0) THEN  (* file opend ok *)
0058        outbuff_addr := ADR(outbuffer);
0059        file_write := TRUE;  (* signal io system to write file *)
0060       file_io_ld; (* write to file using ladder logfc *)
0061        file_write := FALSE;
0062      END_IF
0063      write_recipe_button := FALSE;
0064  END_IF
0065
```

PLC_VISU



trend01



Wago 758- #4 | Switch Scree #1 | %t%H:%M:%S #2 | Trend Data Scree #3

Data_points

```
0001
0002  VAR_GLOBAL
0003  END_VAR
0004
0005  VAR_GLOBAL CONSTANT
0006  END_VAR
0007
0008  VAR_GLOBAL RETAIN
0009  END_VAR
0010
0011  VAR_GLOBAL RETAIN
0012  END_VAR
```

Globale_Variablen

```
0001  VAR_GLOBAL
0002      Observer : BOOL;
0003      Warning : BOOL;
0004      Stop : BOOL;
0005      Start : BOOL;
0006      flash_state : BOOL;
0007      demo_val : WORD;
0008    (*  count01 : BOOL;  *)
0009      default_delay : TIME := t#6000ms;
0010        delay01: TIME := t#6000ms;
0011      delay02: TIME := t#6000ms;
0012      delay01_secs, delay02_secs : WORD := 6; (* delay in whole seconds *)
0013
0014      failed: BOOL;
0015      read_recipe_button : BOOL;
0016      write_recipe_Button : BOOL;
0017      recipe_01 : BOOL;
0018      curr_recipe : INT :=0; (* current recipe in use, start with defaults *)
0019      file_write : BOOL := FALSE;  (* false for file read, true for file write*)
0020  (* globals for read buffer *)
0021      infile: DWORD;   (* input file handle *)
0022      inbuffer: ARRAY[0..9] OF BYTE;
0023      inbuff_addr: DWORD;  (* address of inbuffer *)
0024      readbytes: DWORD := 5;   (* read only 5 bytes, 2 values sep by comma or space *)
0025      file_read_count: DWORD;
0026      read_file_close_err : BOOL;
0027
0028  (* globaks for write buffer *)
0029      outfile: DWORD;
0030      outbuffer: ARRAY[0..9] OF BYTE;
0031      outbuff_addr: DWORD;
0032      writebytes: DWORD := 10;   (* read only 10 bytes,*)
0033      file_write_count: DWORD;
0034      write_file_close_err : BOOL;
0035
0036  (* note: file names are platform specific *)
0037
0038      rec_fname_01 : STRING := 'recipe01.txt';
0039      rec_fname_02 : STRING := 'recipe02.txt';
0040      rec_fname_03 : STRING := '/media/sda1/recipe03.txt';
0041      rec_fname_04 : STRING := '/media/sda1/recipe04.txt';
0042
0043      load_def_recipe : BOOL := FALSE; (* this can be set globally *)
0044
0045      change_visu_button : BOOL := FALSE;
0046      soft_sel_visu : BOOL := FALSE;
0047      cur_visu: WORD := 1;
0048      new_visu: WORD := 1;
```

```
0049
0050      del01u : BOOL := FALSE; (* button for delay01 up *)
0051      del01d : BOOL := FALSE; (* button for delay01 down *)
0052      del02u : BOOL := FALSE;  (* button for delay02 up *)
0053      del02d : BOOL := FALSE; (* button for delay02 down *)
0054  END_VAR
```

Input_Output

```
0001
0002  VAR_GLOBAL
0003  END_VAR
0004
0005  VAR_GLOBAL CONSTANT
0006  END_VAR
0007
0008  VAR_GLOBAL RETAIN
0009  END_VAR
```

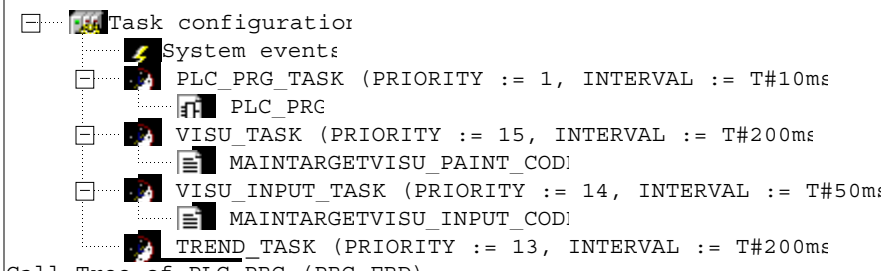
Variablen_Konfiguration

```
0001  VAR_CONFIG
0002  END_VAR
```

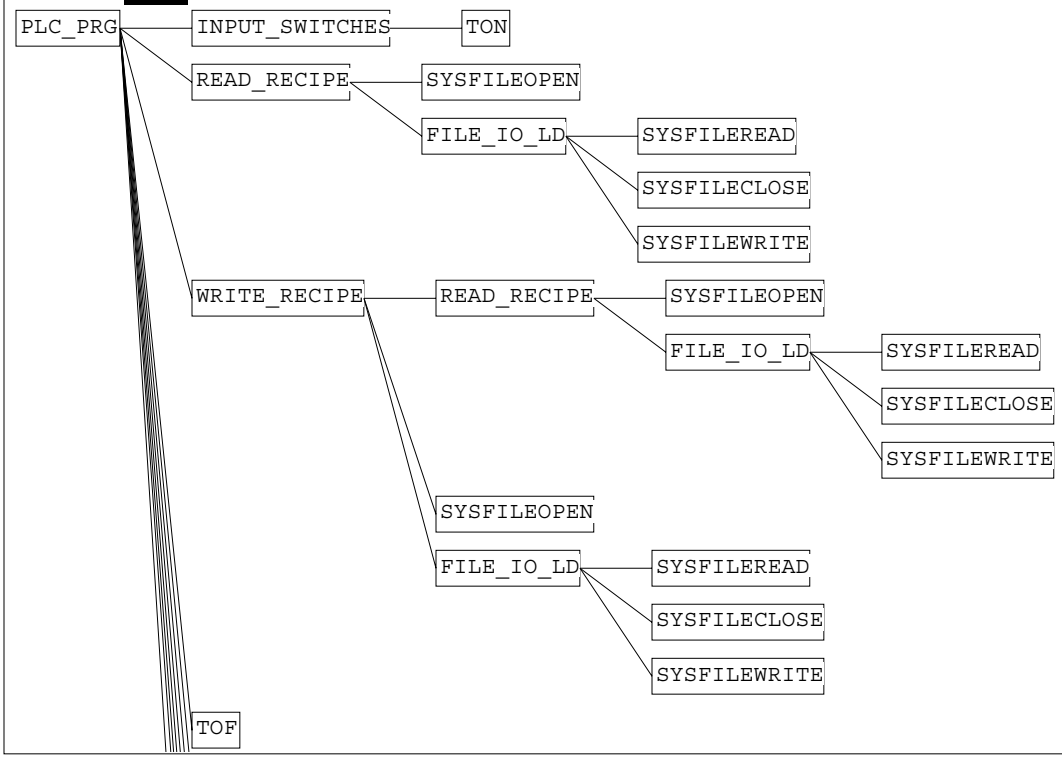
PLC Configuration

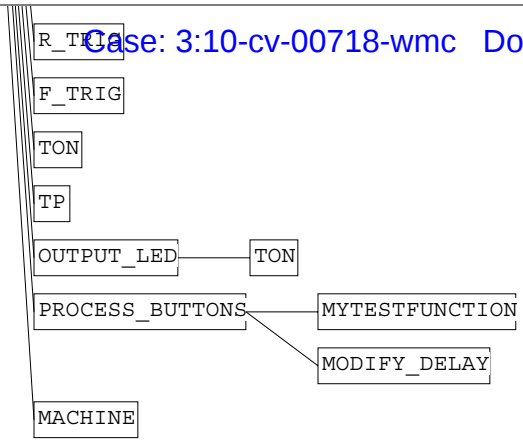
```
*PLC Configuration (Id.: 8765)
  Node number:  -1
  Input address: %IB0
  Output address: %QB0
  Diagnostic address: %MB0
  Download:  1
  AutoAdr:  1

  *K-Bus[FIX] (Id.: 11994)
    Node number:  0
    Input address: %IB0
    Output address: %QB0
    Diagnostic address: %MB0
    Download:  1
    AutoAdr:  1

    *0750-1502 8DI / 8DO 24V DC 0.5A Ribbon Cable[VAR] (Id.: 2000034008)
      Node number:  0
      Input address: %IB0
      Output address: %QB0
      Diagnostic address: %MB0
      Download:  1
      AutoAdr:  1
      Channels:
         AT %IX0.0: BOOL; (* Ch_1 Digital input *) [CHANNEL (I)]
        LED01 AT %QX0.0: BOOL; (* First output *) [CHANNEL (Q)]
         AT %IX0.1: BOOL; (* Ch_2 Digital input *) [CHANNEL (I)]
        LED02 AT %QX0.1: BOOL; (* Ch_2 Digital output *) [CHANNEL (Q)]
         AT %IX0.2: BOOL; (* Ch_3 Digital input *) [CHANNEL (I)]
        LED03 AT %QX0.2: BOOL; (* Ch_3 Digital output *) [CHANNEL (Q)]
         AT %IX0.3: BOOL; (* Ch_4 Digital input *) [CHANNEL (I)]
        LED04 AT %QX0.3: BOOL; (* Ch_4 Digital output *) [CHANNEL (Q)]
         AT %IX0.4: BOOL; (* Ch_5 Digital input *) [CHANNEL (I)]
        LED05 AT %QX0.4: BOOL; (* Ch_5 Digital output *) [CHANNEL (Q)]
         AT %IX0.5: BOOL; (* Ch_6 Digital input *) [CHANNEL (I)]
        LED06 AT %QX0.5: BOOL; (* Ch_6 Digital output *) [CHANNEL (Q)]
         AT %IX0.6: BOOL; (* Ch_7 Digital input *) [CHANNEL (I)]
        LED07 AT %QX0.6: BOOL; (* Ch_7 Digital output *) [CHANNEL (Q)]
         AT %IX0.7: BOOL; (* Ch_8 Digital input *) [CHANNEL (I)]
        LED08 AT %QX0.7: BOOL; (* Ch_8 Digital output *) [CHANNEL (Q)]

  *Internal Digital I/O[FIX] (Id.: 9)
    Node number:  1
    Input address: %IB512
    Output address: %QB512
    Diagnostic address: %MB0
```

```
    Download:  1
    AutoAdr:  1

    *Internal Digital Inputs[FIX] (Id.: 10)
      Node number:  0
      Input address: %IB512
      Output address: %QB512
      Diagnostic address: %MB4
      Download:  1
      AutoAdr:  1
      Channels:
        DIN1 AT %IX2300.0: BOOL; (* Internal Digital Input Bit 1 *) [CHANNEL (I)]
        DIN2 AT %IX2300.1: BOOL; (* Internal Digital Input Bit 2 *) [CHANNEL (I)]

    *Internal Digital Outputs[FIX] (Id.: 11)
      Node number:  1
      Input address: %IB513
      Output address: %QB512
      Diagnostic address: %MB8
      Download:  1
      AutoAdr:  1
      Channels:
        DOUT1 AT %QX2300.0: BOOL; (* Internal Digital Output Bit 1 *) [CHANNEL (Q)]
        DOUT2 AT %QX2300.1: BOOL; (* Internal Digital Output Bit 2 *) [CHANNEL (Q)]

  *Fieldbus variables[FIX] (Id.: 2010310001)
    Node number:  3
    Input address: %IB0
    Output address: %QB0
    Diagnostic address: %MB0
    Download:  1
    AutoAdr:  1
Task configuration
```



Call Tree of PLC_PRG (PRG-FBD)




R_TRIG
F_TRIG
TON
TP
OUTPUT_LED
TON
PROCESS_BUTTONS
MYTESTFUNCTION
MODIFY_DELAY
MACHINE